National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Awadhesh Trivedi
Student ID: 23222468

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Awadhesh Trivedi ……

Student ID: ……23222468…….……

Programme: ……Ms in Data Analytics…… **Year:** ……2024……..

Module: …Research Project…….……

Lecturer: …Abid Yaqoob…….……

Submission Due Date: …12/12/2024…….……

Project Title: ……Options pricing using Machine Learning……..……

Word Count: ……937…… **Page Count:** ……11……..……..……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Awadhesh……

Date: ……12/12/2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Awadhesh Trivedi
23222468

1 Introduction

This configuration guide can be used to run the code developed for this study. The statement generically implies that the code will run flawlessly and without hitches when the instructions in it are executed sequentially. Minimum specifications and other requirements for code execution and the hardware needs for it are also presented in the document. Implementing these suggestions will enhance reproduction of all project results and promote more complicated study in the future.

2 System Specification

2.1 Hardware Configuration

Below are the required system specifications to execute the code:

- Processor: Intel Core i7
- System Memory: 1TB SSD Hard Disk
- RAM: 16GB

2.2 Software Comfiguration

The software requirements are discussed below:

- Windows Edition: Windows 11
- Integrated Development Environment: Jupyter Notebook
- Scripting Language: Python 3 +
- Storage: Local System Storage

3 Python Libraries

The "pip install Library_Name" command is used to install the Python libraries in the Jupyter Notebook environment.
Below are the libraries used for thesis implementation:

Numpy: For the arithmetic operation and for the array and matrix manipulations and mathematical functions like square root, mean etc.

Pandas: For input/output of datasets as well as for missing observations and other initial steps in data preparation.

Matplotlib.pyplot: A plotting package on which plots like the scatter plot or the line plot needs to be performed to compare actuals with the predicted values.

Seaborn: A statistical visualization library that is employed for plotting specialized kinds of charts such as heat maps to analyze correlation coefficients.

Scikit-learn: Provides tools for machine learning, including:

- LabelEncoder which has been used for encoding the labels for features like 'Survived', 'Sex' and 'Embarked'.
- StandardScaler for scaling the features.
- Basic linear models regression and include Linear regression, decision trees, Random forest, and even Neural networks.
- Some of the measurements that can be used for assessment in the current model include, mean_squared_error and r2_score.

Xgboost: A gradient boosting library which is lightweight and optimized for both speed and performance training of the XGBoost Regressor model.

Catboost: A library to perform gradient boosting for categorical features used in training of Catboost Regressor.

Keras : machine learning API written in python(compatible with tensorflow) Applied when you need to develop deep learning models such as LSTM Neural Networks, when working with sequential data.

Yahoo Finance: To download the option data for analysis

4 Project Development

Code is ready to run after the necessary Python libraries have been installed.

4.1 Data Extraction

```python
import yfinance as yf
import pandas as pd

# Downloading options data for Apple (AAPL)
ticker = "AAPL"
stock = yf.Ticker(ticker)

# Get available expiration dates
expiration_dates = stock.options
print("Available Expiration Dates:", expiration_dates)

# Limit the expiration dates to around 20 (or fewer if there are not enough dates)
num_dates = min(50, len(expiration_dates))  # Use up to 20 expiration dates or fewer if less are available
selected_expiration_dates = expiration_dates[:num_dates]

# Create empty lists to store the calls and puts data for selected expiration dates
all_calls = []
all_puts = []

# Loop through each selected expiration date and get the option chain
for exp_date in selected_expiration_dates:
    print(f"\nFetching options data for expiration date: {exp_date}")

    # Get the option chain for the current expiration date
    option_chain = stock.option_chain(exp_date)

    # Append the calls and puts data to the lists
    all_calls.append(option_chain.calls.assign(Expiration_Date=exp_date))
    all_puts.append(option_chain.puts.assign(Expiration_Date=exp_date))

    # Display the first few rows of the calls and puts data for each expiration date
    print("Calls:\n", option_chain.calls.head())
    print("Puts:\n", option_chain.puts.head())

# Concatenate the results into single DataFrames for calls and puts
calls_df = pd.concat(all_calls, ignore_index=True)
puts_df = pd.concat(all_puts, ignore_index=True)

# Show combined calls and puts data for all selected expiration dates
print("\nCombined Calls Data:\n", calls_df.head())
print("\nCombined Puts Data:\n", puts_df.head())

# Optionally, save the data to CSV files for further analysis
calls_df.to_csv('AAPL_Calls_20_Expirations.csv', index=False)
puts_df.to_csv('AAPL_Puts_20_Expirations.csv', index=False)
```

Data has been taken from Yahoo Finance for analysis using the code snippet above.

Below figure shows the extracted dataframes head

strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	contractSize	currency	Expiration_Date
105.0	121.60	135.95	139.75	0.000000	0.000000	2.0	2	2.093755	True	REGULAR	USD	2024-12-13
140.0	89.06	102.05	103.65	0.000000	0.000000	2.0	5	1.406253	True	REGULAR	USD	2024-12-13
150.0	93.31	92.05	93.60	8.220001	9.660361	38.0	1	2.105473	True	REGULAR	USD	2024-12-13
160.0	83.47	82.05	83.65	-0.180000	-0.215183	3.0	14	1.078130	True	REGULAR	USD	2024-12-13
165.0	58.67	77.95	78.95	0.000000	0.000000	2.0	2	1.673830	True	REGULAR	USD	2024-12-13

4.2 Data preprocessing

Handling Missing Values: The dataset is identified for missing values and addressed.

Removing Duplicates: It will remove duplicate rows and leaves only unique rows.

Feature Scaling and Normalization: To make all numeric features of same order of magnitude, each such feature is scaled with Min-Max scaling or StandardScaler etc. so that model convergence is improved during training.

Feature Engineering: Variables are created from the dataset to increase the dataset's predictive power. For example: Introducing a "Type" variable to tell the difference between 0 for call and 1 for put options.

Handling Multicollinearity: To avoid high redundancies, bid and ask are dropped and to detect multicollinear features, Variance Inflation Factor (VIF) is calculated.

Data Splitting: E.g. 80 - 20 split dataset on which the model is tested on unseen data.

Validation Checks: In order to make sure that the model is generally good, across any other subset of data, you employ cross-validation techniques.

Final Dataset Preparation: Data is saved in clean processed format that ready for modeling, with no errors or inconsistencies in the final dataset.

4.3 Data Visualization

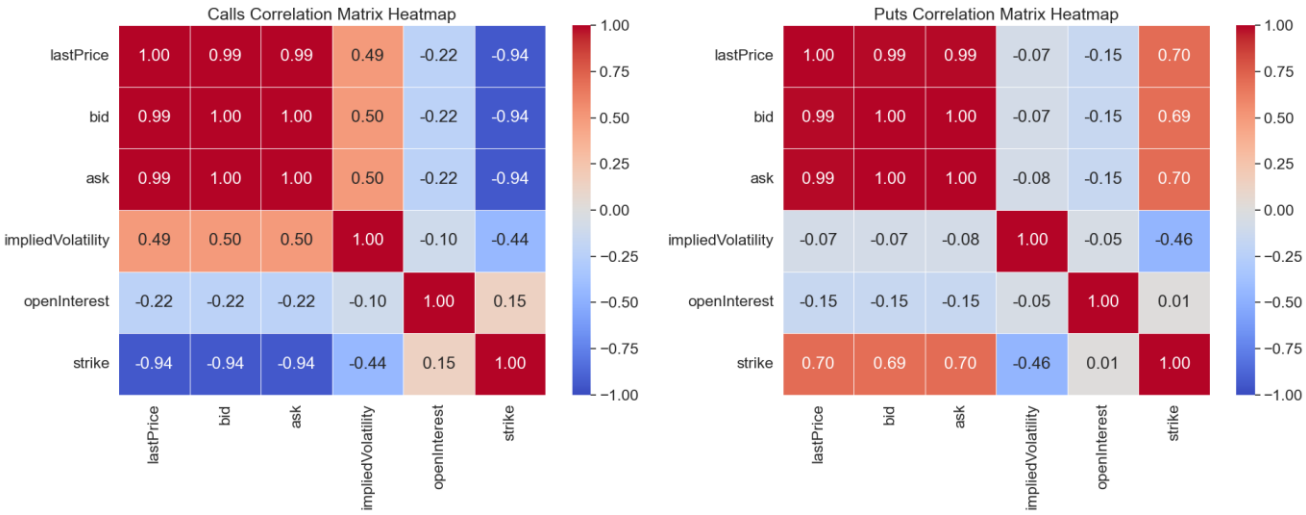

Figure 1. Corelation Heatmap of Numerical variables with strike price

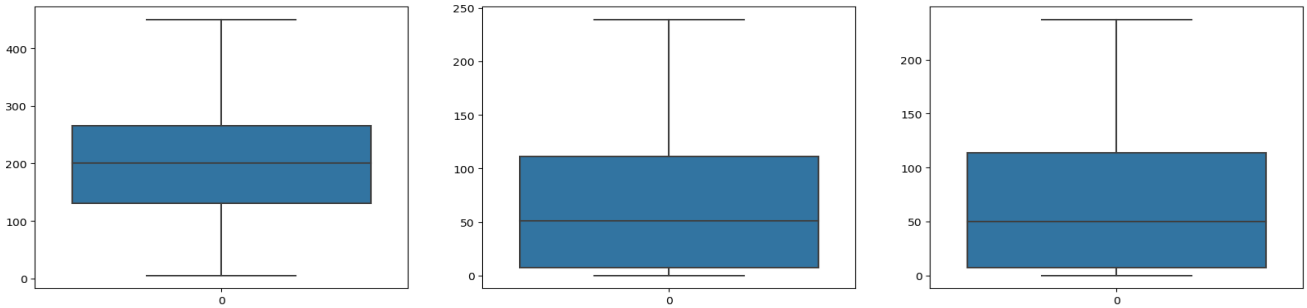

Figure 2. Boxplot of Strike, Last price, and Bid variable (Calls data)

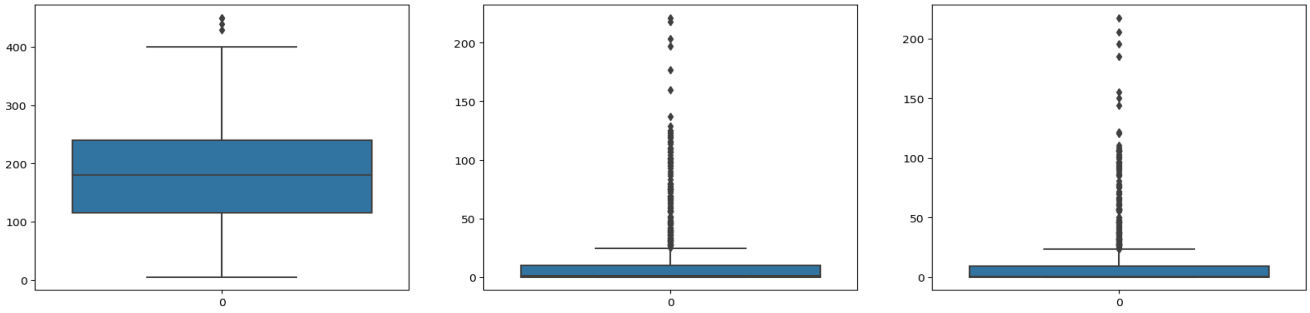

Figure 3. Boxplot of Strike, Last price, and Bid variable (Puts Data)

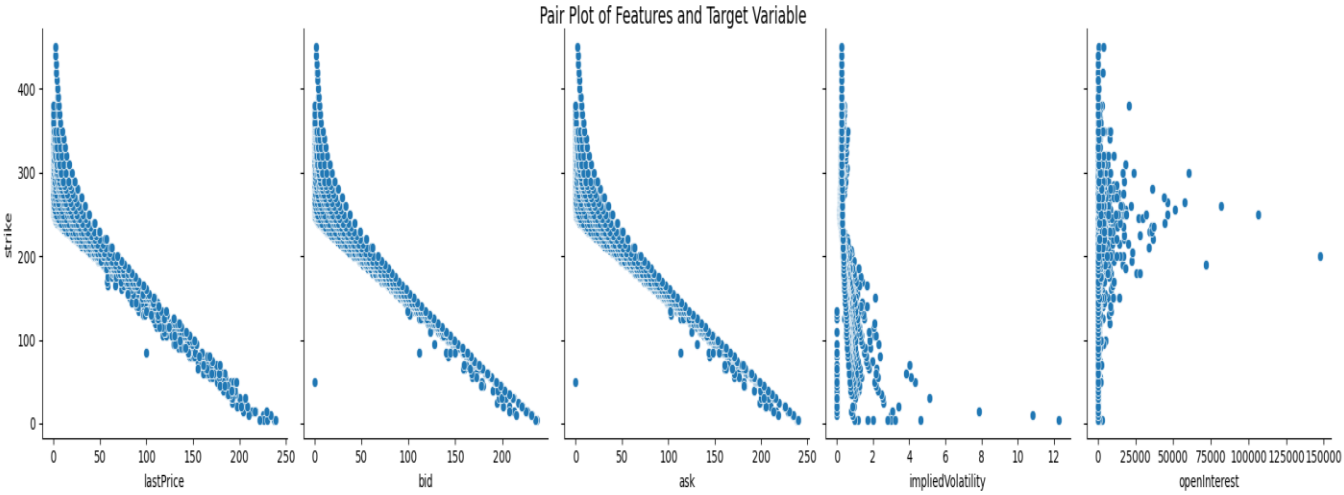

Figure 4. Scatter plot of Calls data

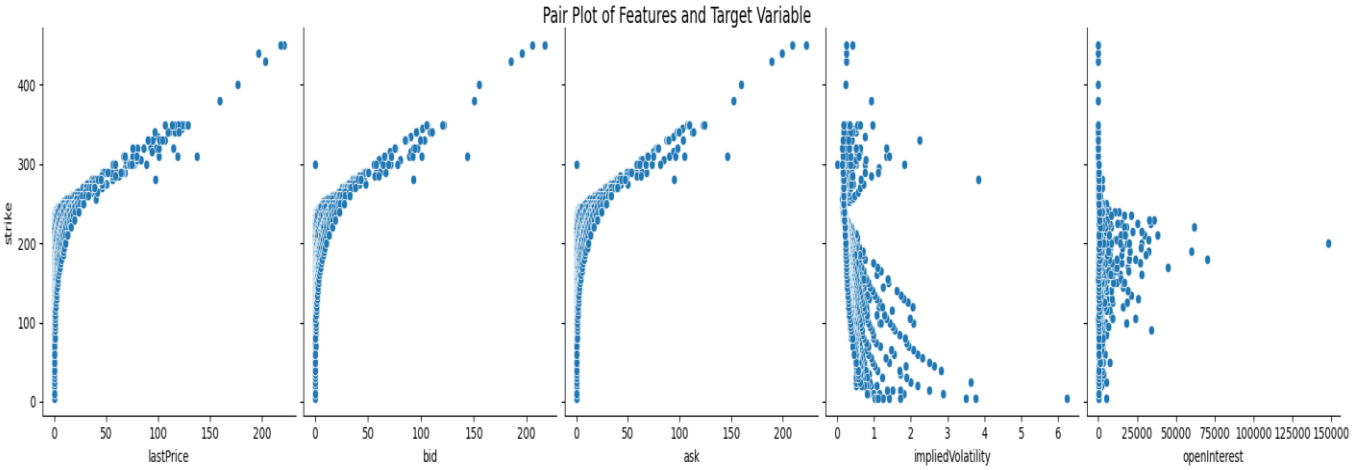

Figure 5. Scatter plot of Puts data

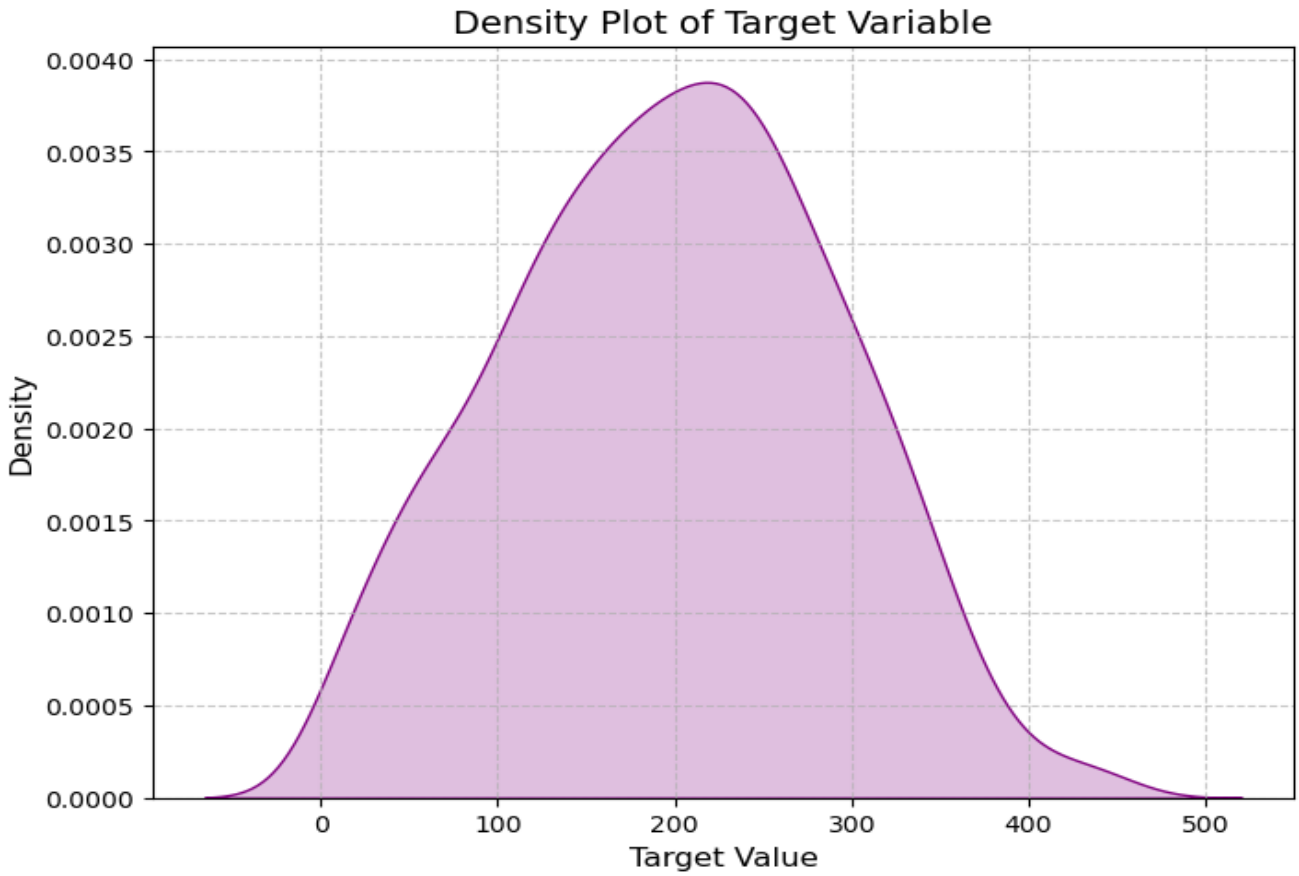

Figure 6. Density plot of target variable

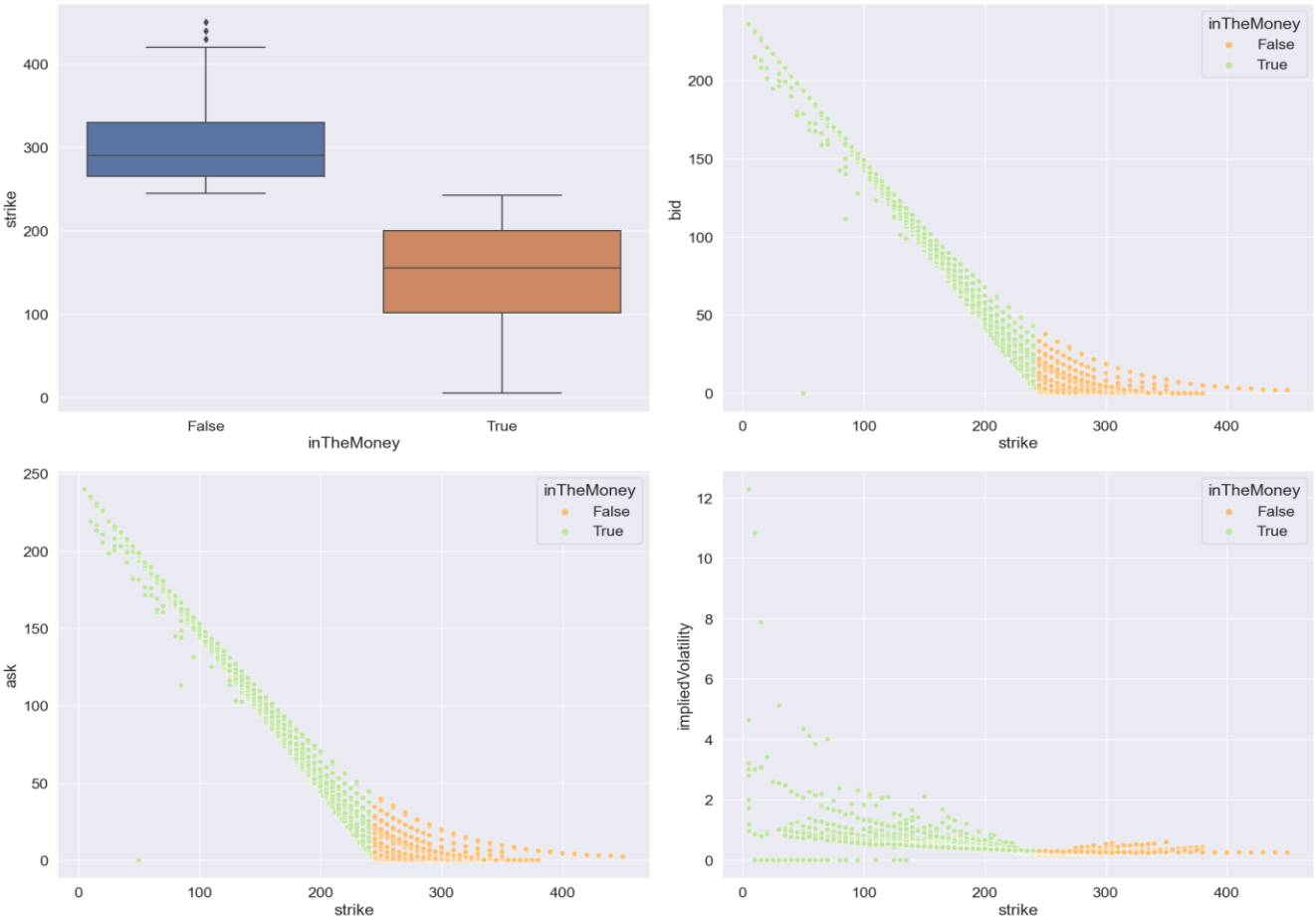

Figure 7. Pair plot of Strike variable

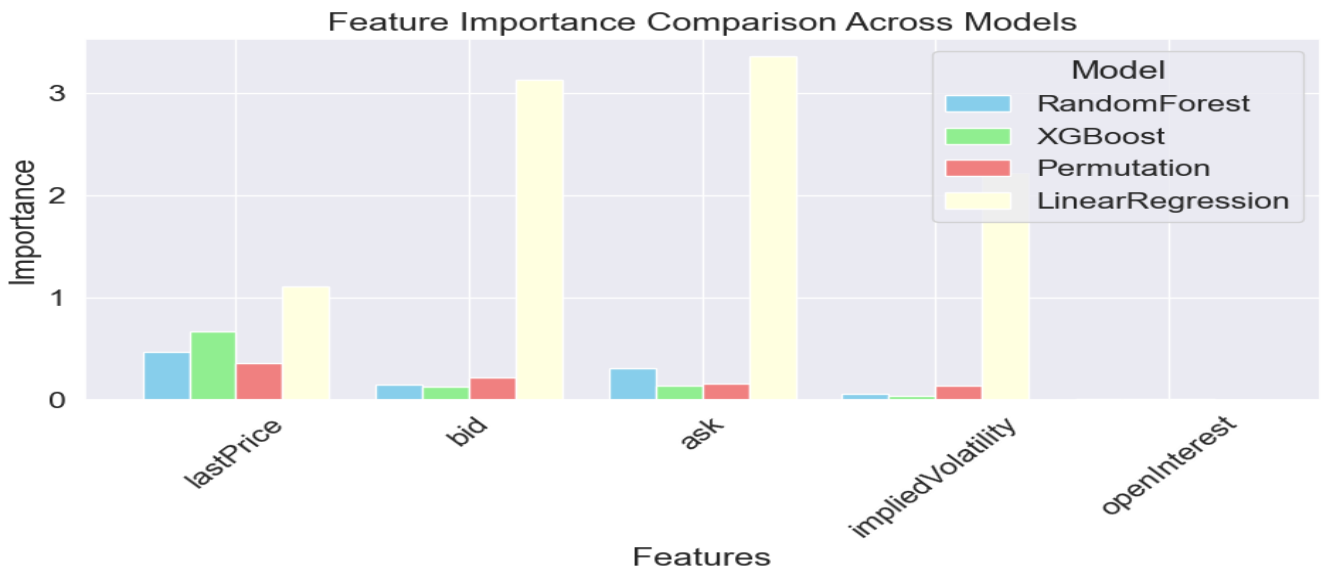

Figure 8. Feature importance bar plot

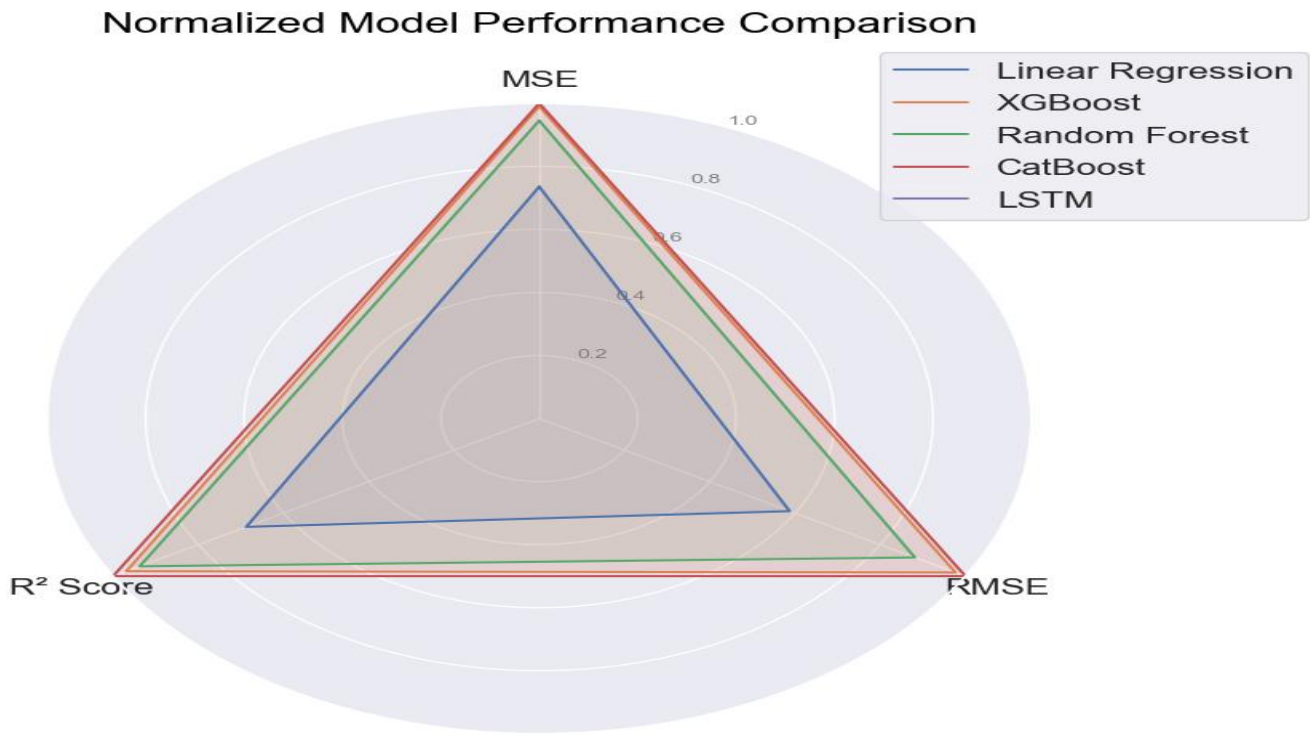

Figure 9. Model performance plot

4.4 Data Splitting & Transformation

```
features = calls_df[['lastPrice', 'bid', 'ask', 'impliedVolatility','openInterest']]
target = calls_df['strike']
```

```
features.isnull().sum()
```

```
lastPrice           0
bid                 0
ask                 0
impliedVolatility   0
openInterest        0
dtype: int64
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

```
# Initialize the StandardScaler
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
# Apply the same transformation to the test data
X_test_scaled = scaler.transform(X_test)
```

Above code is to split the data between train and test and then transform the using standard scaler for transformation

```python
def calculate_vif(dataframe):
    """
    Calculate Variance Inflation Factor (VIF) for each feature in the dataframe.

    Args:
        dataframe (pd.DataFrame): DataFrame containing independent variables.

    Returns:
        pd.DataFrame: DataFrame with features and their corresponding VIF values.
    """
    vif_data = pd.DataFrame()
    vif_data["Feature"] = dataframe.columns
    vif_data["VIF"] = [variance_inflation_factor(dataframe.values, i) for i in range(dataframe.shape[1])]
    return vif_data

features = calls_df[['lastPrice', 'bid', 'ask', 'impliedVolatility', 'openInterest']]
features = features.dropna()

# Calculate VIF
vif_result = calculate_vif(features)
print(vif_result)
```

Above function is to check the varaince inflation factor between the variables

```python
# Example Dataset
features = puts_df[['lastPrice', 'bid', 'ask', 'impliedVolatility', 'openInterest']]
target = puts_df['strike']

# Train a Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(features, target)

# Extract Feature Importance
importance = model.feature_importances_
feature_names = features.columns

# Create a DataFrame for easy visualization
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importance
}).sort_values(by='Importance', ascending=False)

print(importance_df)

# Plot Feature Importance
plt.figure(figsize=(8, 6))
plt.bar(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance')
plt.xticks(rotation=45)
plt.show()
```

The code above is to check the feature importance of the data.
Similarly multiple feature importance tecniques are applied to check the importance of feature

4.5 Model Building

```
# 1. Plain Linear Regression (No Hyperparameters)
linear_model = LinearRegression()
linear_scores = cross_val_score(linear_model, X_train_scaled, y_train, cv=5, scoring='r2')
print("Linear Regression R² Scores:", linear_scores)
print("Average R² for Linear Regression:", linear_scores.mean())

# 2. Ridge Regression Hyperparameter Tuning
ridge_model = Ridge()
ridge_params = {
    'alpha': [0.01, 0.1, 1, 10, 100]  # Regularization strength
}

ridge_grid = GridSearchCV(ridge_model, ridge_params, cv=5, scoring='r2', n_jobs=-1)
ridge_grid.fit(X_train_scaled, y_train)
print("Best Ridge Parameters:", ridge_grid.best_params_)
print("Best Ridge R² Score:", ridge_grid.best_score_)

# 3. Lasso Regression Hyperparameter Tuning
lasso_model = Lasso()
lasso_params = {
    'alpha': [0.01, 0.1, 1, 10, 100]  # Regularization strength
}

lasso_grid = GridSearchCV(lasso_model, lasso_params, cv=5, scoring='r2', n_jobs=-1)
lasso_grid.fit(X_train_scaled, y_train)
print("Best Lasso Parameters:", lasso_grid.best_params_)
print("Best Lasso R² Score:", lasso_grid.best_score_)
```

Hyper-parameter tuned Linear regression

```
# 1. Random Forest Hyperparameter Tuning
rf_model = RandomForestRegressor(random_state=42)
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

rf_grid = GridSearchCV(rf_model, rf_params, cv=5, scoring='r2', n_jobs=-1)
rf_grid.fit(X_train_scaled, y_train)
print("Best Random Forest Parameters:", rf_grid.best_params_)
print("Best Random Forest R² Score:", rf_grid.best_score_)

# 2. XGBoost Hyperparameter Tuning
xgb_model = XGBRegressor(random_state=42, verbosity=0)
xgb_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2]
}

xgb_grid = GridSearchCV(xgb_model, xgb_params, cv=5, scoring='r2', n_jobs=-1)
xgb_grid.fit(X_train_scaled, y_train)
print("Best XGBoost Parameters:", xgb_grid.best_params_)
print("Best XGBoost R² Score:", xgb_grid.best_score_)

# 3. CatBoost Hyperparameter Tuning
cat_model = CatBoostRegressor(random_state=42, verbose=0)
cat_params = {
    'iterations': [100, 200, 300],
    'depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2]
}
```

Hyperparameter tunned RandomForest, Xgboost, and Catboost regressor.

```python
# Prepare data
data = data_m[['lastPrice', 'bid', 'ask', 'impliedVolatility', 'openInterest', 'strike','Type']]

# Sort data (if sequential information is needed)
data = data.sort_index()

# Separate features and target
features = data[['lastPrice','bid', 'ask', 'impliedVolatility', 'openInterest','Type']]
target = data['strike']

# Normalize features and target
scaler_x = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler_x.fit_transform(features)
target_scaled = scaler_y.fit_transform(target.values.reshape(-1, 1))

# Prepare data for LSTM: convert to sequences
def create_sequences(X, y, time_steps=10):
    X_seq, y_seq = [], []
    for i in range(len(X) - time_steps):
        X_seq.append(X[i:i + time_steps])
        y_seq.append(y[i + time_steps])
    return np.array(X_seq), np.array(y_seq)

time_steps = 10
X_seq, y_seq = create_sequences(features_scaled, target_scaled, time_steps)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size=0.2, random_state=42)

# Build LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu', return_sequences=False))
model.add(Dense(1))  # Output layer for regression
model.compile(optimizer='adam', loss='mse')

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

# Evaluate the model
loss = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")

# Make predictions
y_pred = model.predict(X_test)
```

LSTM Model implementation.

Final Model output comparision:

Model	MSE	RMSE	R^2
Linear Regression	4888.51	69.91	0.33
XGBoost	710.04	26.65	0.9
Random Forest	892.9	29.88	0.88
CatBoost	358.89	18.94	0.95
LSTM	4310.96	65.66	0.46